Performance Measurement and Analysis Tools for Cray XE/XK Systems

Heidi Poxon
Cray Inc.

Topics

- Introduction
- Steps to using the Cray performance tools
- Automatic profiling analysis
- Performance Counters

Design Goals

- **Assist the user with application performance analysis and optimization**
  - Help user identify important and meaningful information from potentially massive data sets
  - Help user identify problem areas instead of just reporting data
  - Bring optimization knowledge to a wider set of users

- **Focus on ease of use and intuitive user interfaces**
  - Automatic program instrumentation
  - Automatic analysis

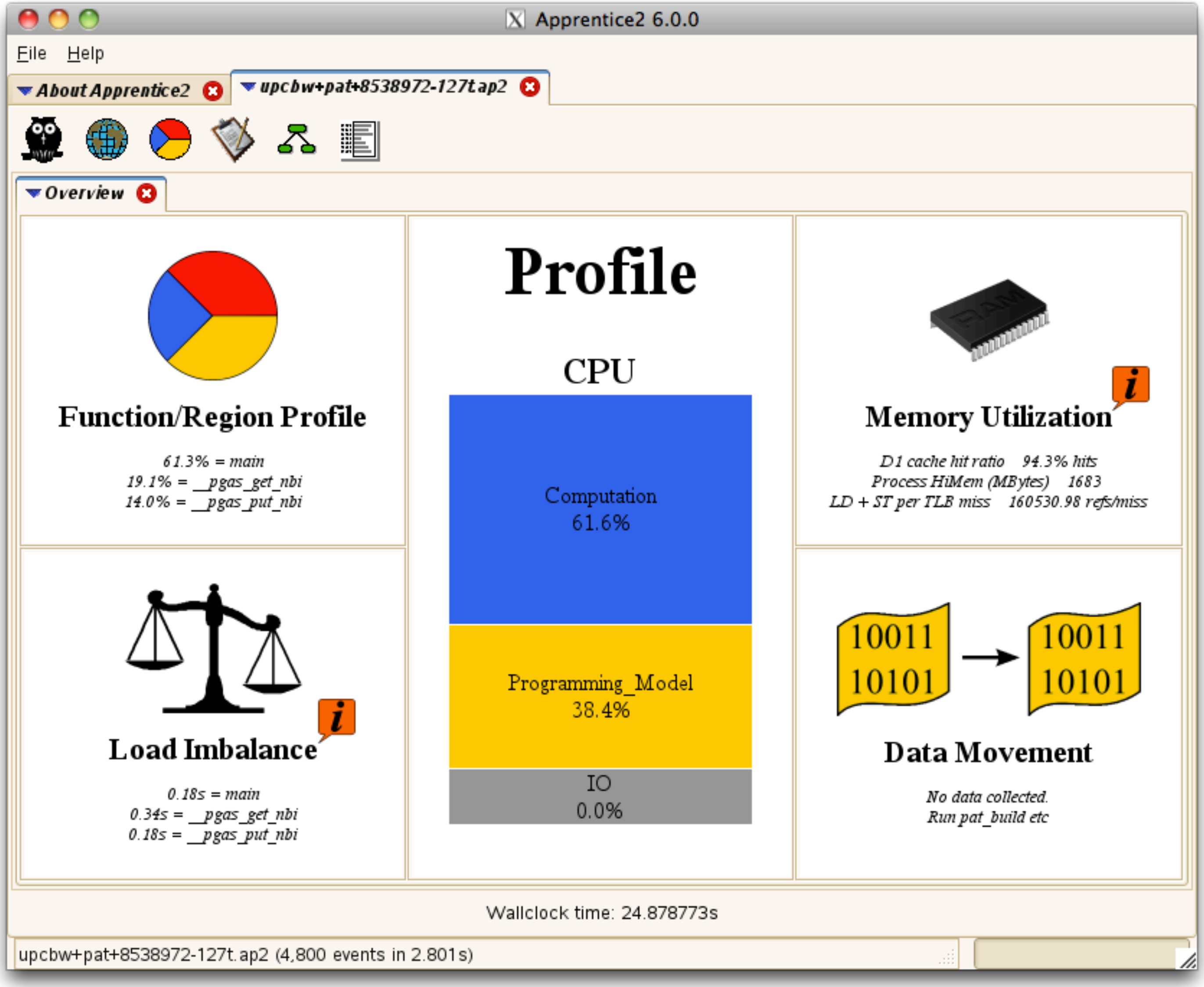
- **Target scalability issues in all areas of tool development**
  - Data management
    - Storage, movement, presentation

Strengths

Provide a complete solution from instrumentation to measurement to analysis to visualization of data

- **Performance measurement and analysis on large systems**
  - Automatic Profiling Analysis
  - Load Imbalance
  - HW counter derived metrics
  - Predefined trace groups provide performance statistics for libraries called by program (blas, lapack, pgas runtime, netcdf, hdf5, etc.)
  - Observations of inefficient performance
  - Data collection and presentation filtering
  - Data correlates to user source (line number info, etc.)
  - Support MPI, SHMEM, OpenMP, UPC, CAF, OpenACC
  - Access to network counters
  - Minimal program perturbation

Application Performance Summary



The Cray Performance Analysis Framework

- **Supports traditional post-mortem performance analysis**
  - Automatic identification of performance problems
    - Indication of causes of problems
    - Suggestions of modifications for performance improvement

  - pat_build: provides automatic instrumentation
  - CrayPat run-time library collects measurements (transparent to the user)
  - pat_report performs analysis and generates text reports
  - pat_help: online help utility
  - Cray Apprentice2: graphical visualization tool

- **To access software:**
  - module load perftools

Application Instrumentation with pat_build

- **pat_build is a stand-alone utility that instruments the application for performance collection**

- **Requires no source code or makefile modification**
  - Automatic instrumentation at group (function) level
    - Groups: mpi, io, heap, math SW, …

- **Performs link-time instrumentation**
  - **Requires object files**
  - Instruments optimized code
  - Generates stand-alone instrumented program
  - Preserves original binary

Application Instrumentation with pat_build (2)

- **Supports two categories of experiments**
  - asynchronous experiments (sampling) which capture values from the call stack or the program counter at specified intervals or when a specified counter overflows
  - Event-based experiments (tracing) which count some events such as the number of times a specific system call is executed
- **While tracing provides most useful information, it can be very heavy if the application runs on a large number of cores for a long period of time**
- **Sampling can be useful as a starting point, to provide a first overview of the work distribution**

Sampling with Line Number information

heidi@limited: /h/heidi — ssh — 81×26

```
Table 2:  Profile by Group, Function, and Line

 Samp%  |   Samp  | Imb.   |  Imb.   |Group
        |         | Samp   | Samp%   | Function
        |         |        |         |  Source
        |         |        |         |   Line
        |         |        |         |    PE=HIDE

 100.0% | 8376.9 |     -- |      -- |Total
|-----------------------------------------------------------------------------
|  93.2% | 7804.0 |     -- |      -- |USER
||----------------------------------------------------------------------------
||  51.7% | 4328.7 |     -- |      -- |calc3_
3|        |        |        |         | heidi/DARPA/cache_util/calc3.do300-ijswap.F
||||--------------------------------------------------------------------------
4|||  15.7% | 1314.4 |  93.6 |   6.8% |line.78
4|||  13.9% | 1167.7 |  98.3 |   7.9% |line.79
4|||  14.5% | 1211.6 |  97.4 |   7.6% |line.80
4|||   1.2% |  103.1 |  26.9 |  21.2% |line.93
4|||   1.1% |   88.4 |  22.6 |  20.8% |line.94
4|||   1.0% |   84.5 |  17.5 |  17.6% |line.95
4|||   1.0% |   86.8 |  33.2 |  28.2% |line.96
4|||   1.3% |  105.0 |  23.0 |  18.4% |line.97
4|||   1.4% |  116.5 |  24.5 |  17.7% |line.98
||||==========================================================================
                                                          144,1          38%
```

Where to Run Instrumented Application

- **By default, data files are written to the execution directory**
- **Default behavior requires file system that supports record locking, such as Lustre (/mnt/snx3/… , /lus/…, /scratch/ …,etc.)**
  - Can use PAT_RT_EXPFILE_DIR to point to existing directory that resides on a high-performance file system if not execution directory
- **Number of files used to store raw data**
  - 1 file created for program with 1 – 256 processes
  - $\sqrt{n}$ files created for program with 257 – *n* processes
  - Ability to customize with PAT_RT_EXPFILE_MAX
- **See intro_craypat(1) man page**

CrayPat Runtime Options

- **Runtime controlled through PAT_RT_XXX environment variables**
- **See intro_craypat(1) man page**
- **Examples of control**
  - Enable full trace
  - Change number of data files created
  - Enable collection of HW counters
  - Enable collection of network counters
  - Enable tracing filters to control trace file size (max threads, max call stack depth, etc.)

Example Runtime Environment Variables

- **Optional timeline view of program available**
  - export PAT_RT_SUMMARY=0
  - View trace file with Cray Apprentice^2
- **Number of files used to store raw data:**
  - 1 file created for program with 1 – 256 processes
  - $\sqrt{n}$ files created for program with 257 – *n* processes
  - Ability to customize with PAT_RT_EXPFILE_MAX
- **Request hardware performance counter information:**
  - export PAT_RT_HWPC=<HWPC Group>
  - Can specify events or predefined groups

pat_report

- Combines information from binary with raw performance data
- Performs analysis on data
- Generates text report of performance results
- Generates customized instrumentation template for automatic profiling analysis
- Formats data for input into Cray Apprentice^2

Why Should I generate a ".ap2" file?

- The ".ap2" file is a self contained compressed performance file
- Normally it is about 5 times smaller than the ".xf" file
- Contains the information needed from the application binary
  - Can be reused, even if the application binary is no longer available or if it was rebuilt
- It is the only input format accepted by Cray Apprentice^2

Files Generated and the Naming Convention

File Suffix	Description
a.out+pat	Program instrumented for data collection
a.out…s.xf	Raw data for sampling experiment, available after application execution
a.out…t.xf	Raw data for trace (summarized or full) experiment, available after application execution
a.out…st.ap2	Processed data, generated by pat_report, contains application symbol information
a.out…s.apa	Automatic profiling pnalysis template, generated by pat_report (based on pat_build –O apa experiment)
a.out+apa	Program instrumented using .apa file
MPICH_RANK_ORDER.Custom	Rank reorder file generated by pat_report from automatic grid detection an reorder suggestions

Automatic Profiling Analysis

Program Instrumentation - Automatic Profiling Analysis

- **Automatic profiling analysis (APA)**
  - Provides simple procedure to instrument and collect performance data for novice users
  - Identifies top time consuming routines
  - Automatically creates instrumentation template customized to application for future in-depth measurement and analysis

Steps to Collecting Performance Data

- **Access performance tools software**

```
% module load perftools
```

- **Build application keeping .o files (CCE: -h keepfiles)**

```
% make clean
% make
```

- **Instrument application for automatic profiling analysis**
  - You should get an instrumented program a.out+pat

```
% pat_build -O apa a.out
```

- **Run application to get top time consuming routines**
  - You should get a performance file (“<sdatafile>.xf”) or multiple files in a directory <sdatadir>

```
% aprun … a.out+pat   (or  qsub <pat script>)
```

Steps to Collecting Performance Data (2)

- **Generate report and .apa instrumentation file**

  ```
  % pat_report <sdatafile>.xf > my_sampling_report
  ```

  Or

  ```
  % pat_report -o my_sampling_report [<sdatafile>.xf | <sdatadir>]
  ```

- **Inspect .apa file and sampling report**
- **Verify if additional instrumentation is needed**

APA File Example

```
#  You can edit this file, if desired, and use it
#   to reinstrument the program for  tracing like this:
#
#          pat_build -O standard.cray-
     xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2-Oapa.512.quad.cores.seal.
     090405.1154.mpi.pat_rt_exp=default.pat_rt_hwpc=none.14999.xf.xf.apa
#
#  These suggested trace options are based on data from:
#
#    /home/users/malice/pat/Runs/Runs.seal.pat5001.2009Apr04/./pat.quad/
     homme/standard.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2-Oapa.
     512.quad.cores.seal.
     090405.1154.mpi.pat_rt_exp=default.pat_rt_hwpc=none.14999.xf.xf.cdb
# ----------------------------------------------------------------------

#       HWPC group to collect by default.

  -Drtenv=PAT_RT_HWPC=1  # Summary with TLB metrics.

# ----------------------------------------------------------------------

#       Libraries to trace.

  -g mpi

# ----------------------------------------------------------------------

#       User-defined functions to trace, sorted by % of samples.

#       The way these functions are filtered can be controlled with
#       pat_report options (values used for this file are shown):
#
#       -s apa_max_count=200    No more than 200 functions are listed.
#       -s apa_min_size=800     Commented out if text size < 800 bytes.
#       -s apa_min_pct=1        Commented out if it had < 1% of samples.
#       -s apa_max_cum_pct=90   Commented out after cumulative 90%.

#       Local functions are listed for completeness, but cannot be traced.

  -w  # Enable tracing of user-defined functions.
      # Note: -u should NOT be specified as an additional option.
```

```
# 31.29%  38517 bytes
         -T prim_advance_mod_preq_advance_exp_

# 15.07%  14158 bytes
         -T prim_si_mod_prim_diffusion_

#  9.76%  5474 bytes
         -T derivative_mod_gradient_str_nonstag_

. . .

#  2.95%  3067 bytes
         -T forcing_mod_apply_forcing_

#  2.93%  118585 bytes
         -T column_model_mod_applycolumnmodel_

#  Functions below this point account for less than 10% of samples.

#  0.66%  4575 bytes
#          -T bndry_mod_bndry_exchangev_thsave_time_

#  0.10%  46797 bytes
#          -T baroclinic_inst_mod_binst_init_state_

#  0.04%  62214 bytes
#          -T prim_state_mod_prim_printstate_

. . .
#  0.00%  118 bytes
#          -T time_mod_timelevel_update_

# ----------------------------------------------------------------------

  -o preqx.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2.x+apa
     # New instrumented program.

  /.AUTO/cray/css.pe_tools/malice/craypat/build/pat/2009Apr03/2.1.56HD/
     amd64/homme/pgi/pat-5.0.0.2/homme/2005Dec08/build.Linux/preqx.cray-
     xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2.x  # Original program.
```

Generating Profile from APA

- Instrument application for further analysis (a.out+apa)

```
% pat_build -O <apafile>.apa
```

- Run application

```
% aprun … a.out+apa   (or  qsub <apa script>)
```

- Generate text report and visualization file (.ap2)

```
% pat_report -o my_text_report.txt [<datafile>.xf |
   <datadir>]
```

- View report in text and/or with Cray Apprentice[2]

```
% app2 <datafile>.ap2
```

Program Instrumentation Tips

- **Large programs**
  - Scaling issues more dominant
  - Use automatic profiling analysis to quickly identify top time consuming routines
  - Use loop statistics to quickly identify top time consuming loops
- **Small (test) or short running programs**
  - Scaling issues not significant
  - Can skip first sampling experiment and directly generate profile
  - For example: % pat_build -u -g upc my_program

Example Experiments

- **> pat_build –O apa**
  - Gets you top time consuming routines
  - Lightest-weight sampling
- **> pat_build –u –g mpi ./my_program**
  - Collects information about user functions and MPI
- **> pat_build –w ./my_program**
  - Collects information for MAIN
  - Lightest-weight tracing
- **> pat_build –gnetcdf,mpi ./my_program**
  - Collects information about netcdf routines and MPI

Predefined Trace Wrappers (-g tracegroup)

blas	Basic Linear Algebra subprograms
caf	Co-Array Fortran (Cray CCE compiler only)
hdf5	manages extremely large data collection
heap	dynamic heap
io	includes stdio and sysio groups
lapack	Linear Algebra Package
math	ANSI math
mpi	MPI
omp	OpenMP API
pthreads	POSIX threads
shmem	SHMEM
sysio	I/O system calls
system	system calls
upc	Unified Parallel C (Cray CCE compiler only)

For a full list, please see pat_build(1) man page

Specific Tables in pat_report

```
heidi@kaibab:/lus/scratch/heidi> pat_report -O -h

pat_report: Help for -O option:
Available option values are in left column, a prefix can be
specified:
   ct                          -O calltree
   defaults                    <Tables that would appear by default.>
   heap                        -O heap_program,heap_hiwater,heap_leaks
   io                          -O read_stats,write_stats
   lb                          -O load_balance
   load_balance                -O lb_program,lb_group,lb_function
   mpi                         -O mpi_callers
   ---
   D1_D2_observation           Observation about Functions with low D1+D2
cache hit ratio
   D1_D2_util                  Functions with low D1+D2 cache hit ratio
   D1_observation              Observation about Functions with low D1
cache hit ratio
   D1_util                     Functions with low D1 cache hit ratio
   TLB_observation             Observation about Functions with low TLB
refs/miss
   TLB_util                    Functions with low TLB refs/miss
```

CrayPat API - For Fine Grain Instrumentation

- **Fortran**

```
include "pat_apif.h"
…
call PAT_region_begin(id, "label", ierr)
do i = 1,n
…
enddo
call PAT_region_end(id, ierr)
```

- **C & C++**

```
include <pat_api.h>
…
ierr = PAT_region_begin(id, "label");
< code segment >
ierr = PAT_region_end(id);
```

CPU HW Performance Counters

Hardware Performance Counters - IL

- **AMD Family 15H Opteron Hardware Performance Counters**
  - Each node has **4** 48-bit NorthBridge counters
  - Each core has **6** 48-bit performance counters
    - Not all events can be counted on all counters
    - Supports multi-events
      - events have a maximum count per clock that exceeds one event per clock

PAPI Predefined Events

- **Common set of events deemed relevant and useful for application performance tuning**
  - Accesses to the memory hierarchy, cycle and instruction counts, functional units, pipeline status, etc.
  - The “papi_avail” utility shows which predefined events are available on the system – execute on compute node
- **PAPI also provides access to native events**
  - The “papi_native_avail” utility lists all AMD native events available on the system – execute on compute node
- **PAPI uses perf_events Linux subsystem**
- **Information on PAPI and AMD native events**
  - pat_help counters
  - man intro_papi (points to PAPI documentation: http://icl.cs.utk.edu/papi/)
  - http://lists.eecs.utk.edu/pipermail/perfapi-devel/2011-January/004078.html

Hardware Counters Selection

- **HW counter collection enabled with PAT_RT_HWPC environment variable**
- **PAT_RT_HWPC <set number> | <event list>**
  - A set number can be used to select a group of predefined hardware counters events (recommended)
    - CrayPat provides 23 groups on the Cray XT/XE systems
    - See pat_help(1) or the hwpc(5) man page for a list of groups
  - Alternatively a list of hardware performance counter event names can be used
  - Hardware counter events are not collected by default

HW Counter Information Available in Reports

- Raw data
- Derived metrics
- Desirable thresholds

Predefined Interlagos HW Counter Groups

See pat_help -> counters -> amd_fam15h –> groups

- 0: Summary with instructions metrics
- 1: Summary with TLB metrics
- 2: L1 and L2 Metrics
- 3: Bandwidth information
- 4: <Unused>
- 5: Floating operations dispatched
- 6: Cycles stalled, resources idle
- 7: Cycles stalled, resources full
- 8: Instructions and branches
- 9: Instruction cache
- 10: Cache Hierarchy (unsupported for IL)

Predefined Interlagos HW Counter Groups (cont'd)

11: Floating point operations dispatched
12: Dual pipe floating point operations dispatched
13: Floating point operations SP
14: Floating point operations DP
19: Prefetchs
20: FP, D1, TLB, MIPS
21: FP, D1, TLB, Stalls
22: D1, TLB, MemBW
23: FP, D1, D2, and TLB
default: group 23

Support for L3 cache counters coming in 3Q2013

New HW counter groups for Interlagos (6 counters)

- **Group 20: FP, D1, TLB, MIPS**
  - PAPI_FP_OPS
  - PAPI_L1_DCA
  - PAPI_L1_DCM
  - PAPI_TLB_DM
  - DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  - PAPI_TOT_INS
- **Group 21: FP, D1, TLB, Stalls**
  - PAPI_FP_OPS
  - PAPI_L1_DCA
  - PAPI_L1_DCM
  - PAPI_TLB_DM
  - DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  - PAPI_RES_STL

Example: HW counter data and Derived Metrics

```
  PAPI_TLB_DM  Data translation lookaside buffer misses
  PAPI_L1_DCA  Level 1 data cache accesses
  PAPI_FP_OPS  Floating point operations
  DC_MISS      Data Cache Miss
  User_Cycles  Virtual Cycles
========================================================================
USER
------------------------------------------------------------------------
  Time%                                          98.3%
  Time                                       4.434402 secs
  Imb.Time                                         -- secs
  Imb.Time%                                        --
  Calls                         0.001M/sec     4500.0 calls
  PAPI_L1_DCM                  14.820M/sec   65712197 misses
  PAPI_TLB_DM                   0.902M/sec    3998928 misses
  PAPI_L1_DCA                 333.331M/sec 1477996162 refs
  PAPI_FP_OPS                 445.571M/sec 1975672594 ops
  User time (approx)            4.434 secs 11971868993 cycles  100.0%Time
  Average Time per Call                      0.000985 sec
  CrayPat Overhead : Time        0.1%
  HW FP Ops / User time       445.571M/sec 1975672594 ops   4.1%peak(DP)
  HW FP Ops / WCT             445.533M/sec
  Computational intensity        0.17 ops/cycle    1.34 ops/ref
  MFLOPS (aggregate)          1782.28M/sec
  TLB utilization              369.60 refs/miss   0.722 avg uses
  D1 cache hit,miss ratios      95.6% hits         4.4% misses
  D1 cache utilization (misses) 22.49 refs/miss   2.811 avg hits
========================================================================
```

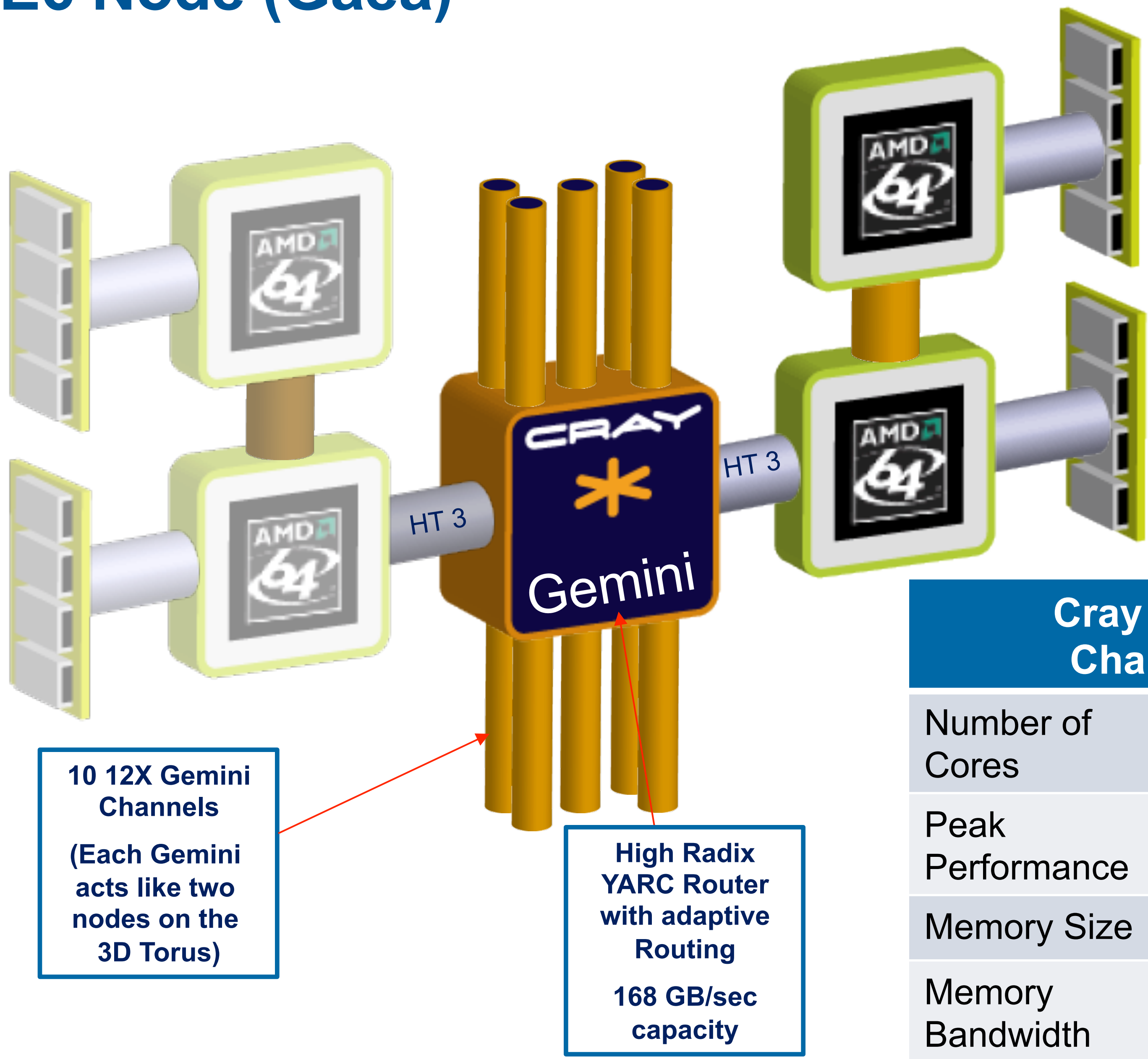
PAT_RT_HWPC=1
Flat profile data
Raw counts
Derived metrics

PAT_RT_HWPC=2 (L1 and L2 Metrics)

```
========================================================================
USER
------------------------------------------------------------------------
  Time%                                                  98.3%
  Time                                               4.436808 secs
  Imb.Time                                                 -- secs
  Imb.Time%                                                --
  Calls                          0.001M/sec            4500.0 calls
  DATA_CACHE_REFILLS:
    L2_MODIFIED:L2_OWNED:
    L2_EXCLUSIVE:L2_SHARED       9.821M/sec          43567825 fills
  DATA_CACHE_REFILLS_FROM_SYSTEM:
    ALL                         24.743M/sec         109771658 fills
  PAPI_L1_DCM                   14.824M/sec          65765949 misses
  PAPI_L1_DCA                  332.960M/sec        1477145402 refs
  User time (approx)             4.436 secs       11978286133 cycles  100.0%Time
  Average Time per Call                              0.000986 sec
  CrayPat Overhead : Time         0.1%
  D1 cache hit,miss ratios       95.5% hits              4.5% misses
  D1 cache utilization (misses)  22.46 refs/miss        2.808 avg hits
  D1 cache utilization (refills)  9.63 refs/refill      1.204 avg uses
  D2 cache hit,miss ratio        28.4% hits             71.6% misses
  D1+D2 cache hit,miss ratio     96.8% hits              3.2% misses
  D1+D2 cache utilization        31.38 refs/miss        3.922 avg hits
  System to D1 refill           24.743M/sec         109771658 lines
  System to D1 bandwidth      1510.217MB/sec       7025386144 bytes
  D2 to D1 bandwidth           599.398MB/sec       2788340816 bytes
========================================================================
```

Gemini Network Performance Counters

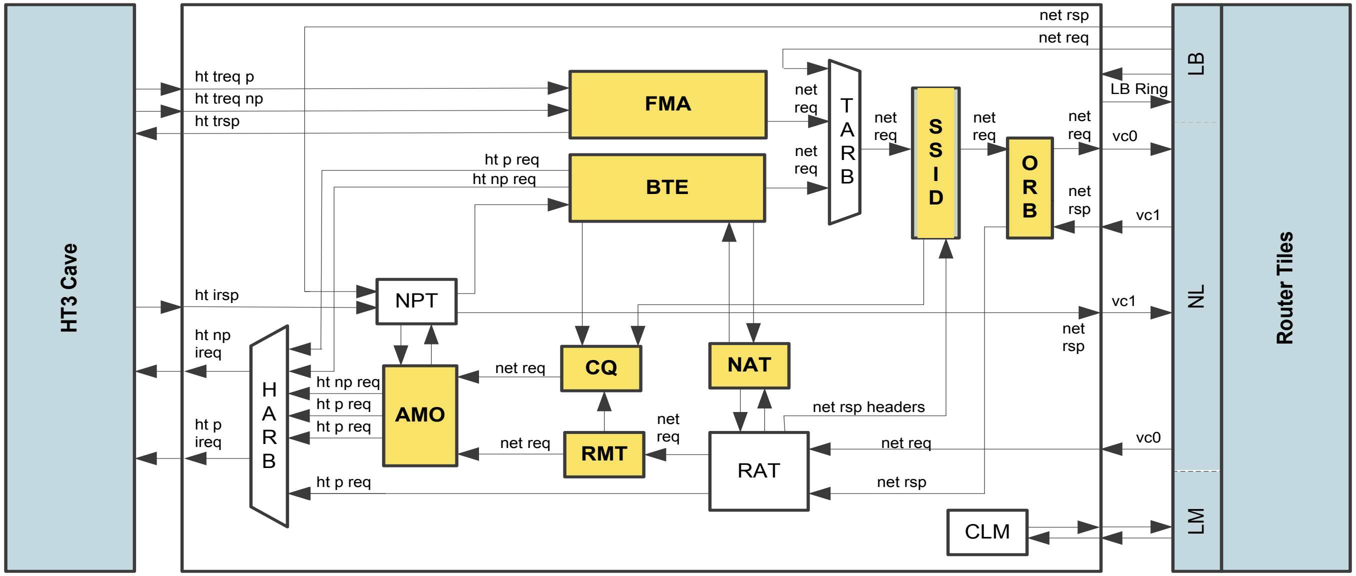
XE6 Node (Gaea)



Cray Baker Node Characteristics	
Number of Cores	32*
Peak Performance	~300 Gflops/s
Memory Size	64 GB per node
Memory Bandwidth	85 GB/sec

Gemini Network Interface

- Fast Memory Access (FMA) – fine grain remote PUT/GET
- Block Transfer Engine (BTE) – offload for long transfers
- Completion Queue (CQ) – client notification
- Atomic Memory Op (AMO) – fetch&add, etc.



Overview

- **2 categories of performance counters**
  - **NIC** – record information about data moving through the Network Interface Controller
    - 2 NICs per Gemini ASIC, each attached to a compute node
    - Counters reflect network transfers beginning and ending on the node
    - Easy to associate with an application
    - Each NIC connects to a different node, running a separate OS instance
  - **Router tiles** –
    - Available on a per-Gemini basis
    - 48 router tiles, arranged in 6x8 grid
    - 8 processor tiles connect to each of the two NICs (called PTILEs)
      - Data is associated with any traffic from the 2 nodes connected to the Gemini
    - 40 network tiles (NTILEs) connect to the other Gemini's on the system
      - Data is associated with any traffic passing through the router (not necessarily from your application)

Using the Tools to Monitor Gemini Counters

- Network counter events are not collected by default
- Access to counter information is expensive (on the order of 2 us for 1 counter)
- We suggest you do not collect any other performance data when collecting network counters as they can skew the non-counter results
- When collecting counters, ALPS will not place a different job on the same Gemini (the second node)

Using the Tools to Monitor Gemini Counters (2)

- Network counter collection enabled with PAT_RT_NWPC environment variable
- PAT_RT_NWPC <event list> | <file containing event list>
- See the nwpc(5) man page for a list of groups
- See the intro_craypat(1) man page for environment variables that enable network counters
- See "Using the Cray Gemini Hardware Counters" available at http://docs.cray.com

How to Collect Network Statistics

- **Instrument program for tracing:**

  ```
  $ pat_build -w my_program
  ```

- **Enable and choose network counter collection:**

  ```
  $ export PAT_RT_NWPC=GM_ORB_PERF_VC0_STALLED
  ```

- **Run program:**

  ```
  $ aprun my_program+pat
  ```

Example Default Gemini Counter Output

```
Notes for table 2:
  Table option:
    -O profile_nwpc
  Options implied by table option:
    -d ti%@0.95,ti,N -b gr,fu,ni=HIDE -s show_data=rows

  The Total value for each data item is the sum for the Group values.
  The Group value for each data item is the sum for the Function values.
  The Function value for each data item is the avg for the Node Id values.
    (To specify different aggregations, see: pat_help report options s1)

  This table shows only lines with Time% > 0.95.
    (To set thresholds to zero, specify:  -T)

  Percentages at each level are of the Total for the program.
    (For percentages relative to next level up, specify:
      -s percent=r[elative])

Table 2:  NWPC Data by Function Group and Function
Group / Function / Node Id=HIDE
==================================================
  Total
--------------------------------------------------
  Time%                                  100.0%
  Time                               405.190432 secs
  GM_TILE_PERF_VC0_PHIT_CNT:0:0      1668962112
  GM_TILE_PERF_VC1_PHIT_CNT:0:0       156579492
  GM_TILE_PERF_VC0_PKT_CNT:0:0         52400892
  GM_TILE_PERF_VC1_PKT_CNT:0:0         52193128
```

Other Views of Network Counter Data

- By default, counter totals are provided
- Can view counters per NID
- Mesh coordinates for job available as of perftools/6.0.0
  - Can look at counters along the X, Y, or Z coordinates
- Can generate csv file to plot data

Other Views of Network Counter Data

- **Can generate csv file to plot data:**

  $ pat_report -s content=tables -s show_data=csv \
  -s notes=hide =s sort_by_pe=yes -d N -b pe

- **What does this mean?...**

- **-s content=tables**
  - Only include table data (exclude job and environment information)
- **-s show_data=csv**
  - Dump data in csv format
- **-s notes=hide**
  - Don’t include table notes in output
- **-s sort_by_pe=yes**
  - Sort data by PE
- **-d N**
  - Display all available network events (1 per column)
- **-b pe**
  - Display each entry in table by PE

Example Counters

Are the routers used by your program congested because of your program or because of other traffic on the system?

- **Ratio of the change in stall counters to the change in sum of phit counters**

- **The following counters are on a per Gemini router tile basis (48 tiles per Gemini) * 3 counters per tile:**
  - GM_TILE_PERF_VC0_PHIT_CNT
  - GM_TILE_PERF_VC1_PHIT_CNT
  - GM_TILE_PERF_INQ_STALL

- Degree of congestion =

GM_TILE_PERF_INQ_STALL / (GM_TILE_PERF_VC0_PHIT_CNT + GM_TILE_PERF_VC1_PHIT_CNT)

Interpreting Counters

- Including network counters in application performance analysis is newer territory for users
- Experimentation is needed to find and understand the most helpful counters
- Goal is to use our tools infrastructure (derived metrics, and performance analysis) to help interpret counters
- Focus of the Cray performance tools is to provide feedback that developers can act on to improve the performance of their program
- We are investigating counters to suggest to users
- User feedback on helpful counters is welcome

Cray PAPI Network Component

- Coming in March 2013
- Available for 3rd party tool developers
- Used internally by CrayPat
- Counter events documented through papi_native_avail

Questions
?